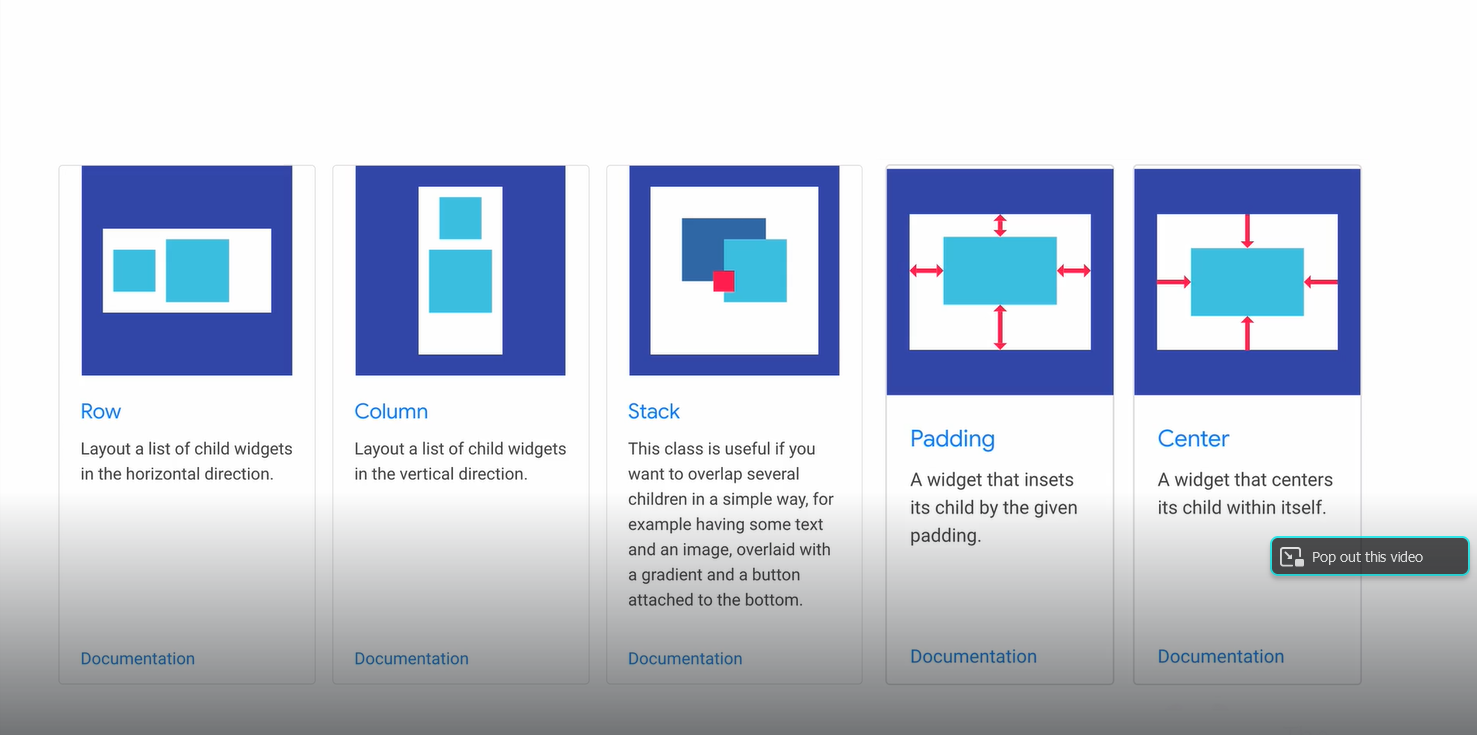
What is flutter?

Flutter is an open-source framework developed by Google for building beautiful, natively compiled, multi-platform applications from a single codebase.

It was officially announced in May 2017 and first stable release, Flutter 1.0, was launched on December 4, 2018.

What are widgets?

In Flutter, a widget is a basic building block of the user interface. Widgets define how a part of the app should look and behave. They can represent anything from a button or a text label to complex layouts or entire screens. Widgets in Flutter are immutable and can be composed together to create more complex UIs.



**Note**: Android and iOS it will just provide a blank windows or canvas however all the drawing and design can be done via widgets from the flutter.

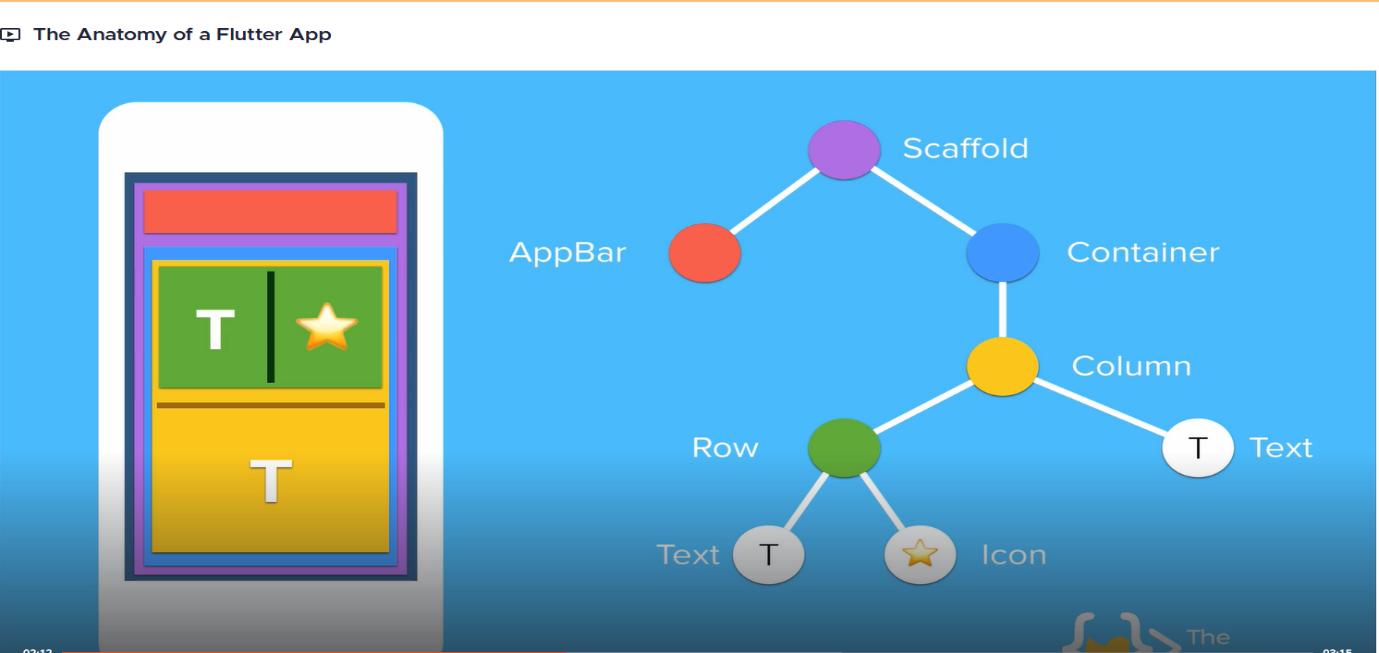
**Note:** In flutter, widgets are building block and by plugin together it builds an appealing application.

|  |  |  |
| --- | --- | --- |
| **Sl.no** | **Stateless Widget** | **Stateful Widget** |
| 1 | if a widget doesn’t do anything called Stateless Widget. | If a widget does anything then called stateful widget. |
| 2 | Static in nature which mean if a developer once code and leave it will be as it forever till developer didn’t change. | Dynamic in nature it is opposite to the stateless widgets. |
| 3 | Users can’t interact | Users can interact |
| 4 | They don’t store any real-time data | They store real-time data and perform tasks accordingly. |
| 5 | Example: Simple Text widgets and Icons, etc. | Example: Button, Slider, checkbox, Radio and TextField etc. |

**Hot Reload**: This allows you to see changes in your code almost instantly without losing the current app state. For example, if you update the UI or fix a bug, you can use hot reload to apply those changes while keeping the app running, so you don’t have to navigate back to the same screen or lose your current data.

**Hot Restart**: This restarts the app from scratch, losing the current app state but quickly reloading the new code. It’s useful for applying changes that affect the app’s initialization or global state, which hot reload doesn’t handle.

**Anatomy of the flutter App**



What is BuildContext?

BuildContext is a locator that is used to track each widget in a tree and locate them and their position in the tree. The BuildContext of each widget is passed to their build method. Remember that the build method returns the widget tree a widget renders.

Each BuildContext is unique to a widget. This means that the BuildContext of a widget is not the same as the BuildContext of the widgets returned by the widget.

**Widget build(BuildContext context)**

1. Purpose: Describes the UI of a widget.
2. Parameters: `BuildContext context` provides information about the widget's location and access to the widget tree.
3. Function: Returns a `Widget` that defines the visual appearance and is called when the widget's state changes or when the widget is first created.
4. Whenever Flutter needs to update the UI (e.g., when the state changes), it calls the build method again to get the updated widget tree.

Differences between the widgets hierarchy and the element and render
trees

**Widgets Tree ()**

In Flutter, the Widget Tree is a hierarchical structure of immutable objects that describes the layout and appearance of the user interface. Each widget in the tree represents a part of the UI, like a button or text, and the tree defines how these parts are organized and nested.

**Elements Tree**

A tree that represents the current state of the widgets and manages their lifecycle. Elements act as intermediaries between the widgets and the render objects.

**Render Tree**

In Flutter, the Render Tree is a hierarchy of `Render Object` instances responsible for the actual layout and painting of the user interface. It takes the layout constraints from its parent, calculates positions and sizes, and draws the UI elements on the screen.

What is runApp()?

The `runApp()` function in Flutter initializes the app by taking a `Widget` as its argument and inflating it to display the app's user interface on the screen. It serves as the entry point for the Flutter app and attaches the given widget to the screen.

What is MyApp()?

In Flutter, `MyApp()` is a user-defined class that typically extends `StatelessWidget` or `StatefulWidget`. It serves as the root widget of the application, defining the app's overall structure and theme. This class is usually passed to the `runApp()` function to set up the app's initial interface.

Sure, here’s a simplified version:

What is a MaterialApp()?

MaterialApp is a predefined class or widget in a flutter. It is likely the main or core component of a flutter app. The MaterialApp widget provides a wrapper around other Material Widgets. We can access all the other components and widgets provided by Flutter SDK.

What is a scaffold()?

**Scaffold** is a class in [**flutter**](https://www.geeksforgeeks.org/flutter-an-introduction-to-the-open-source-sdk-by-google/) which provides many widgets or we can say [APIs](https://www.geeksforgeeks.org/introduction-to-apis/) like Drawer, Snack-Bar, Bottom-Navigation-Bar, Floating-Action-Button, App-Bar, etc. **Scaffold** will expand or occupy the whole device screen. It will occupy the available space. Scaffold will provide a [framework](https://www.geeksforgeeks.org/software-framework-vs-library/) to implement the basic material design layout of the application.

Example:

Scaffold({

Key key,

this.appBar,

this.body,

}

What is a container() ?

The Container widget in Flutter is a versatile widget used for creating and customizing boxes with various properties such as padding, margin, alignment, decoration (like background color or border), and constraints. It's one of the most commonly used widgets in Flutter for laying out UI elements.

What is an AppBar()?

An AppBar is a widget that displays a material design app bar at the top of the screen. It typically contains elements like the title of the app, navigation icons, and actions (e.g., buttons). It's commonly used within the Scaffold widget as part of the app's visual layout.

***AppBar*** is usually the topmost component of the app (or sometimes the bottom-most), it contains the toolbar and some other common action buttons. As all the components in a flutter application are a widget or a combination of widgets.

Widgets are the central class hierarchy in the Flutter framework. They represent an immutable description of part of a user interface. Purpose of the Widgets is described what their view should look like based on their current configuration and state of widgets.

A custom widget created by developers to encapsulate specific UI components or functionality. **Purpose of the** User-defined widgets allow you to compose complex UIs by combining simpler widgets.

setState()

In Flutter, the setState() function is a fundamental part of managing state within a StatefulWidget. It allows you to update the UI by triggering a rebuild of the widget.

What is a State?

The State is the information that can be read synchronously when the widget is built and might change during the lifetime of the widget.

**StatelessWidget:**

The widgets whose state cannot be altered once they are built are called stateless widgets. These widgets are immutable once they are built.

**StatefulWidget:**

The widgets whose state can be altered once they are built are called stateful Widgets. These states are mutable and can be changed multiple times in their lifetime. This simply means the state of an app can change multiple times.

In Flutter, when you create a StatefulWidget, you need to define two classes:

1. The StatefulWidget itself (demo in your case).
2. The State object that holds the state for that StatefulWidget (HomePage in your case).

**Holding State**:

By extending State<demo>, the HomePage class can hold mutable state that can change over time.

**Building the UI**:

The build method in HomePage describes how the UI should be rendered based on the current state. It is called whenever the state changes and setState is invoked.

**Responding to State Changes**:

setState is a method provided by the State class that notifies Flutter to rebuild the UI when the state changes.

**Flutter Project folder structure**

1. **Dart Tool folder:**

.dart\_tool is a folder that Flutter use, but you are not going to do anything in this folder.

1). Package Configurations: Stores configuration files related to the packages used in your project.

2). Flutter Version 3.22.0.

1. **Idea folder:**

**.idea** includes the configuration for Android Studio, it will generate automatically when we use flutter program in Android Studio.

The .idea folder contains IDE-specific settings and configurations for managing your Flutter project, such as module configurations, workspace settings, and VCS integration.

1. **Android folder:**

As the name suggests, this folder contains all the Android related files and code for the application and similar to iOS as well.

1. **Lib (Library) folder**

This is the main folder where we have to write all our application code.

1. **Test folder:**

As the name suggests this folder is used to store and manage testing code for the application.

1. **Web folder**

Web folder contains necessary files to run the flutter application in web browser.

1. **Gitignore file:**

.gitignoreThis is a hidden file IDE used to store the list of files which needs to be ingored when the source code is uploaded/checked into to any Git versioning system like [Github](https://github.com/" \t "_blank) or [Bitbucket](https://bitbucket.org/).

1. **Metadata file:**

metadata is managed by Flutter automatically and this also is a hidden file used by IDEs to track the properties

of the Flutter project.

1. **pubspec.lock**

pubspec.lock is generated based on pudspec.yaml. If you delete this file you can generate it again by getting packages from the next file.

1. **pubspec.yaml (yet another markup language)**  
   This is the only file in all these files in which we have to make changes when we have to use any third-party [flutter package](https://pub.dartlang.org/flutter/). This file is used by public package manager to get and load the packages used in the project.
2. **analysis\_options.yaml**

In a Flutter project, the analysis\_options.yaml file is used to configure Dart's static analysis tool. It helps you enforce coding standards, style guidelines, and rules for the Dart code in your Flutter project.

1. **android.iml**This file also contains Android configuration information about the project which is used by IntelliJ engine.
2. **Build.gradle folder**

This folder is not available at the project creation however when we run a code it will automatically generate the folder. It will store all the generated code after compilation of application.